下载APP

# 09｜扩展搜索：如何快速找到想要的文件?

2021-03-02 尹会生

Python自动化办公实战课 进入课程 >

**讲述：尹会生**
时长 17:38 大小 16.15M

你好，我是尹会生。

工作时间越久，你就会发现，自己电脑里存储的办公文件也越来越多。一旦需要紧急找到某个文件，你又恰巧忘记了文件被存放的具体位置，就只能通过 Windows 的搜索功能对文件名称进行搜索，不过你大概率需要长时间等 Windows 的搜索结果。

之所以查找文件的时间过长，主要有两个原因。

1. 搜索范围太大。搜索过程中可能包含了大量的操作系统或应用软件的缓存文件等无效路径，导致搜索时间过长。

2. 受到硬盘文件数量过多的影响。硬盘的文件数量越多，完整搜索一遍所有文件的时间就越长。

那有没有办法提高搜索的效率，快速找到你想要的文件呢？基于以上两种原因，相应的，我们可以在 Python 中采用指定搜索路径范围和提前建立文件索引的两种方案实现文件的搜索功能。

这两种方案都是基于 Python 的基本搜索方法实现的，因此我先来带你学习一下如何用 Python 的 pathlib 库实现基础文件的搜索。

## 基础搜索方法：用 pathlib 库搜索文件

用 Python 搜索文件时需要使用 pathlib 库的 glob() 函数和 rglob() 函数，glob() 函数可以实现基于文件名的搜索方法，rglob 函数可以实现基于扩展名的搜索方法。

我先来给你看一下 Python 实现基础搜索功能的代码，通过代码来为你讲解一下 Python 是如何搜索到一个文件的：

复制代码

```
from pathlib import Path

base_dir = '/Users/edz/Desktop/'
keywords = '**/*BBC*'

# 遍历base_dir指向的目录下所有的文件
p = Path(base_dir)

# 当前目录下包含BBC的所有文件名称
files = p.glob(keywords)
# files的类型是迭代器
# 通过list()函数转换为列表输出
# print(list(files))

# xlsx结尾的文件
files2 = p.rglob('*.xlsx')
print(list(files2))

# 遍历子目录和所有文件
files3 = p.glob('**/*')
print(list(files3))
```

在这段代码中，我实现了 Python 的基础搜索功能。由于搜索的优化要基于 glob() 函数和 rglob() 函数进行，为了能更好的提升 Python 的搜索的效率，我来带你逐一分析一下 glob() 函数和 rglob() 函数的参数和返回值。

**首先我来带你看一下 glob() 函数和它的参数**, 由于 glob() 进行匹配的是文件的路径和文件名称方式，如: "c:\somepath\to\filename_include_BBC_voice.exe" , 而我们进行文件搜索的时候一般会使用关键字，如“BBC”，因此在搜索时我们需要为关键字加上通配符的形式，如“*BBC*” 。

**通配符是类似正则表达式的元字符的一种特殊符号**，它不能用在正则表达式中，只能用在 glob（全称 global）匹配模式中。

我将 glob() 和 rglob() 函数常用的通配符，以及它们的功能整理成一个表格，供你学习和参考。

| 符号 | 含义 |
| --- | --- |
| * | 匹配0个或多个字符 |
| ? | 匹配一个字符 |
| [] | 匹配置顶范围内的字符，如[0–9]匹配数字 |
| ** | 匹配零个或多个目录及子目录。不包含 . 以及 .. 开头的 |

通过表格我希望你能掌握**如何将通配符和要搜索的关键字进行组合**。比如说，通过使用“**/*BBC*”的方式，我就可以搜索到 Path() 函数指定目录下所有包含 BBC 关键字的文件名。

**接下来我再来带你学习一下 rglob 函数和它的参数**。rglob 函数是从文件路径末尾向前进行匹配的，这是它和 glob() 函数的主要区别, 基于 rglob() 函数的搜索顺序特点，经常被我们用于进行扩展名的搜索，比如说采用 rglob('*.xlsx') 就可以搜索所有的 xlsx 扩展名文件，要比使用 glob() 编写的模式匹配更简单，参数的含义也更清晰。

**最后我再来带你看一下 glob() 和 rglob() 函数的返回值**，有一点我需要提醒你：它们的执行结果是我们之前课程中没有接触过的一种新的数据类型，这种类型叫做“迭代器”。

顾名思义，迭代器的特点是这个数据类型可以支持迭代操作，执行一次 glob() 或 rglob() 函数只返回一个结果。要想得到迭代器所以的值可以使用两种方法。

一种是使用 list() 函数将迭代器转换为我们所熟知的列表数据类型，例如我在列表中就是用了“list(files3)”方式将迭代器转换为了列表。

还有一种方式是使用 for 循环的方式对迭代器的值进行逐一处理。

这两种用法不但可以获取 glob() 返回值，今后我们遇到的迭代器都可以使用 list() 函数和 for 循环取得它的全部值。

通过对 glob() 和 rglob() 函数的参数和返回值的学习，相信你已经掌握了使用功能 Python 搜索文件的基础方法，接下来我来带你通过指定搜索路径和建立索引文件提高搜索的效率。

## 提升搜索效率的两种方法

用 Python 的 pathlib 库实现文件搜索，只是在灵活性上比 Windows 默认的搜索更好，但是搜索效率上并不能带来任何提高。为了减少搜索的等待时间, 接下来，我就教你使用指定搜索路径和建立索引文件两个方法, 提高 pathlib 库的搜索效率。

### 指定搜索路径

我们先来看第一种，指定搜索路径。我们需要通过三个步骤实现：

1. 先生成配置文件，把要搜索的路径写入到配置文件中；
2. 再编写读取配置文件和搜索的自定义函数，把配置文件中的路径读取出来，逐个目录搜索；
3. 最后，将多个目录的搜索结果合并输出，便于你通过结果快速找到自己想要的文件。

先说第一步，怎么使用 Python 读取配置文件。以往我们会把要搜索的路径写入到变量，并把定义路径的变量名称放在代码前几行的位置，便于下次修改搜索目录的时候找到这个

变量。但是对于代码工程稍微复杂的程序来说，往往会有多个代码文件，仍然不利于每次搜索的时候进行搜索路径的修改。

现在我教你一个新的方法，就是把变量放入到一个单独的文件中，这个文件被称作该代码的配置文件。这种方法的好处是你修改搜索目录时不用打开代码文件。假设你的朋友也需要类似功能，那你就可以把代码和配置文件一起发给他，哪怕他完全不会 Python，也能使用你编写的程序实现高效搜索。

那怎么给 Python 脚本增加配置文件呢？

配置文件一般为文本文件。配置文件的格式，一般由软件作者基于软件的功能和自己的习惯来指定，不过也有通用的配置文件格式。比如在 Windows 系统中，最常见的配置文件是扩展名为.ini 的文件，在今天这节课，我们就把.ini 文件格式作为配置文件的标准格式。

.ini 文件格式包含三个部分，分别是节、参数和注释。格式如下：

复制代码

```
节
[section]
参数
 (键=值)
   name=value
注释
注释使用“;”分号表示。在分号后面的文字，直到该行结尾都全部为注解。
;注释内容
```

基于.ini 文件的格式，我把配置搜索路径的配置文件修改为如下：

复制代码

```
[work]
;工作文件保存路径
searchpath=/Users/edz,/tmp

[game]
;娱乐文件保存路径
searchpath=/games,/movies,/music
```

在这段代码中，我设置了 work 和 game 两个“节”，分别代表工作和娱乐。这样设置的好处是，我可以根据不同的用途来搜索不同的目录。如果搜索时使用了更少的目录，也会相应减少搜索的等待时间。

另外，你会发现两个“节”中的参数我都指定成相同的名字 --searchpath，这样设置的好处是我将搜索范围从“工作”改为“娱乐”时，只需要在代码里修改搜索的“节”，不用修改搜索的参数。

除了“节”和“参数”，在配置文件中，你还应该关注我对参数 searchpath 设置值的方式，它的值是我想要进行搜索的路径范围，为了在程序中能够更方便得读取多个路径，我使用逗号来分隔多个路径。

**在编写好配置文件之后，下一步就是取得.ini 文件的完整路径**，我把.ini 文件和脚本放在了相同的目录，我在第七讲为你详细剖析过，怎样通过 __file__ 变量和 pathlib 库相结合，实现读取和脚本在同一目录的文件操作，我就不多说了，如果你不太记得了，可以回去复习下。

找到 search.ini 文件完整路径之后，接下来需要读取并分析.ini 文件格式，Python 有实现这个功能的的库，它叫做 configparser 库，通过这个库你可以直接读取.ini 文件中的 searchpath 参数，不用通过 read() 函数读取文件内容，手动编写分析.ini 文件的脚本了。

接下来，我来带你看一下 Python 读取.ini 文件的代码，帮你理解 Python 是怎样通过配置文件来实现在多个路径搜索文件功能的。

复制代码

```
import configparser
import pathlib
from pathlib import Path

def read_dirs(ini_filename, section, arg):
    """
    通过ini文件名,节和参数取得要操作的多个目录
    """
    current_path = pathlib.PurePath(__file__).parent
    inifile = current_path.joinpath(ini_filename)

    # cf是类ConfigParser的实例
    cf = configparser.ConfigParser()

```

```
15      # 读取.ini文件
16      cf.read(inifile)
17
18      # 读取work节 和 searchpath参数
19      return cf.get(section, arg).split(",")
20
21  def locate_file(base_dir, keywords):
22      p = Path(base_dir)
23      files = p.glob(keywords)
24      return list(files)
25
26
27  dirs = read_dirs('search.ini', 'work', 'searchpath')
28  # ['/Users/edz', '/tmp']
29  keywords = '**/*BBC*'
30
31  # 定义存放查找结果的列表
32  result = []
33
34  # 从每个文件夹中搜索文件
35  for dir in dirs:
36      files = locate_file(dir, keywords)
37      result += files
38
39  # 将PosixPath转为字符串
40  print( [str(r) for r in result] )
```

在这段代码中，读取配置文件和搜索这两个最主要的功能由两个自定义函数实现，它们分别是 read_dirs() 函数和 locate_file() 函数。

read_dirs() 函数实现了读取.ini 文件，并将返回的多个路径处理为列表类型。列表类型适合多组并列的数据，多个目录刚好可以使用列表这种数据类型来存放要搜索的目录名称。

locate_file() 函数通过代码的第 35 行循环功能，对每个目录进行了搜索，并将搜索的结果存入 result 变量。result 变量是一个列表数据类型，由于搜索到的文件可能包含多个匹配的文件路径，我需要将搜索到的结果依次存入 result 列表中，再继续搜索下一个目录，继续通过 append() 函数将结果放入列表，直到所有的目录搜索完成，整个搜索的程序才真正执行结束。

最后还有一点需要你注意，在进行路径处理的过程中，pathlib 库为了规避不同操作系统路径写法的差异，就把路径统一定义为 PosixPath() 对象。因此，你在使用这些路径的时候，需要先将 PosixPath 对象转换为字符串类型。我在代码最后一行通过 Python 内置函数 str() 函数把 PosixPath 对象逐个转换为字符串类型，并再次存入到列表当中。

通过限制在指定的目录搜索这个功能，我们就可以规避在搜索时错误匹配系统文件和软件缓存文件而导致的查找过慢问题。

不过如果我们指定的目录仍然有非常多文件的话，那用这个方法查找起来依然会很慢。别担心，接下来我就教你一种利用空间换时间的方法，提高对指定目录文件搜索效率的方法。

## 建立索引文件

什么是“空间换时间”呢？我来解释一下。

我们知道，文件越多，搜索的时间就越长，因为搜索的时间是随着文件数量呈线性增长的。就像一本书越厚，你从第一页读到最后一页的时间就越长一样。那怎样能快速搜索到书中的内容呢？你会想到图书都有目录功能，通过目录可以加快你找到你想看的图书内容的速度。

对于操作系统来说，一个文件也是由文件名称、大小、文件内容等多个部分组成的，搜索文件的功能只需要文件名称就行了，不需要其他的部分。因此，我们可以在硬盘中新开辟一块空间，将所有的文件名提前存储下来作为文件的索引，下次再查找的时候直接查找索引文件，就能得到搜索结果，而不必再通过硬盘查找真实的文件了。这就是“空间换时间”。

既然索引文件可以加快搜索速度，那我们去**建立索引文件**就可以了。

建立索引文件不需要你重新再写新的程序，我们可以基于指定搜索路径的程序进行改造：

先把配置文件目录下所有文件路径的保存方式**由列表改为文件**；

再把搜索功能改为**从文件搜索**。

我把改造后的代码写出来供你参考。

复制代码

```
def locate_file(base_dir, keywords='**/*'):
    """
    迭代目录下所有文件
    """
```

```
    p = Path(base_dir)
    return p.glob(keywords)

def write_to_db():
    """
    写入索引文件
    """
    current_path = pathlib.PurePath(__file__).parent
    dbfile = current_path.joinpath("search.db")

    with open(dbfile, 'w', encoding='utf-8') as f:
        for r in result:
            f.write(f"{str(r)}\n")

# 读取配置文件
dirs = read_dirs('search.ini', 'work', 'searchpath')

# 遍历目录
result = []
for dir in dirs:
    for files in locate_file(dir):
        result.append(files)

# 将目录写入索引文件
write_to_db()

```

在代码中我增加了 write_to_db() 函数，它在代码的第 16-18 行，我通过写入文件方式替代了写入列表的功能。同时，为了能遍历所有的目录，我还修改了 locate_file() 函数的第二个参数，将它改为“keywords='**/*'”。**通过这两处的修改，就把所有文件路径全部保存到 search.db 文件中了。**

search.db 的文件内容如下，这里记录了配置文件指定的所有目录下的所有文件路径：

复制代码

```
/tmp/com.apple.launchd.kZENgZTtVz
/tmp/com.google.Keystone
/tmp/mysql.sock
/tmp/com.adobe.AdobeIPCBroker.ctrl-edz
/tmp/com.apple.launchd.kZENgZTtVz/Listeners
/tmp/com.google.Keystone/.keystone_install_lock
... ...
```

接下来，我再把搜索功能从列表搜索改造为**从文件搜索**，为了避免每次搜索要重新生成一次 search.db 文件，我要编写一个新的脚本，专门用于从文本中搜索关键字，并将搜索的结果显示出来。

相信你一定想到了我在上一讲为你讲解的正则表达式功能，通过 re.search() 函数刚好可以实现文本的搜索。下面的 locate.py 脚本文件就是我使用正则表达式实现的文本搜索功能：

复制代码

```
import pathlib
import re

keyword = "apple"

# 获取索引文件路径
current_path = pathlib.PurePath(__file__).parent
dbfile = current_path.joinpath("search.db")

# 在索引文件中搜索关键字
with open(dbfile, encoding='utf-8') as f:
    for line in f.readlines():
        if re.search(keyword, line):
            print(line.rstrip())

```

在代码中我利用正则表达式的 re.search() 搜索函数，以 keyword 变量作为搜索的关键字，对 search.db 索引文件的每一行进行了匹配，最后将符合关键字“apple”的文件路径和名称一起显示在屏幕上。

使用这种方式来搜索文件，要比使用操作系统自带的搜索工具快得多，因为我将原本 Windows 搜索硬盘上的文件所消耗的时间拆分成了两部分。

一部分是 updatedb.py 建立索引的时间；

一部分是从 search.db 索引文件查找关键字的时间。

当搜索等待的时间被提前转换为建立索引的时间后，搜索的效率自然就提高了。

但是请你注意，这种方式建立的索引文件会有时效性的问题，一旦硬盘中的文件被删除、移动或改名，你就得重新建立索引文件。而索引文件保存的文件路径和当前真实的文件并不能一一对应，所以你需要及时更新索引文件。

为了保证索引文件的时效性，你还可以把 updatedb.py 脚本加入到 Windows 的开机启动脚本中，每次打开计算机就会自动更新索引文件。通过这种方式就可以让索引文件里的文件路径更加准确。

## 总结

最后，我来为你总结一下。这节课，我为你讲解了如何使用 pathlib 库搜索文件，以及如何使用配置文件和索引文件加快搜索。

利用索引文件减少等待时间，实际上是将 Windows 的搜索进行了拆分，提前将搜索的路径保存到了索引文件中，从索引文件搜索时，就不用再进行真正的文件查找工作了，这就减少了搜索的等待的时间。

这种搜索方法在服务器领域已经被广泛使用，像 Linux 和 MacOS 操作系统中都存在着利用索引文件的搜索命令，对于服务器上变化频率较低的场景，利用索引文件搜索非常实用。

## 思考题

我给你留一道思考题。如何扩展 locate.py 的搜索功能，实现搜索.jpg 扩展名的文件呢?

欢迎把你的思考和想法分享在留言区，我们一起交流讨论。如果课程帮你解决了一些工作上的问题，也欢迎你把课程分享给你的朋友、同事。

提建议

© 版权归极客邦科技所有，未经许可不得传播售卖。 页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 08 | 正则表达式：如何提高搜索内容的精确度？

下一篇 10 | 按指定顺序给词语排序，提高查找效率

## 精选留言 (3)

写留言

**回到原点**
2021-03-28

```
def search_ext(file_ext):
    with open(dbfile, encoding='utf-8') as f:
        for line in f.readlines():
            if line.rstrip().endswith(file_ext):
                print(line.rstrip())...
```

展开

作者回复: 赞

**琥珀•**
2021-03-14

```
expanded_name = "jpg"

with open(dbfile, encoding='utf-8') as f:
    for line in f.readlines():
        if re.search(f".*{expanded_name}", line, flags=re.IGNORECASE):...
```

展开

**Soul of the Drago...**
2021-03-02

老师，我按照您之前说的，用.py格式的脚本文件运行“current_path = pathlib.PurePath(__file__).parent”这一行代码，为什么还是会出现“NameError: name '__file__' is not defined”这样的报错?

作者回复: file前后是两个“_” 可以先确认一下，或者参考我为这一讲写的演示代码，看下是否有书写错误的问题